สุทธิศักดิ์ พงศ์ธนาพาณิช

เปิดโลกการเข้ารหัสข้อมูลด้วยภาษาซี ตอน การบีบอัดข้อมูล ด้วยวิธี Sliding Window (ตอนจบ)

อธิบายถึงการทำงานของโค้ด ในส่วนของการถอดรหัสข้อมูล ซึ่งเป็นการทำงานในส่วนของรูทีน ExpandFile

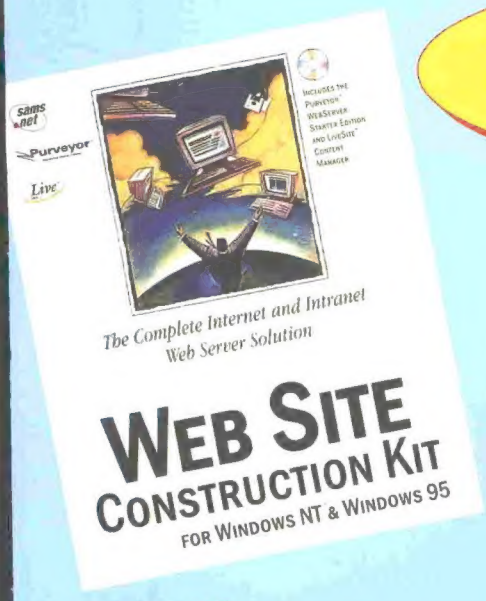
ในบทความตอนที่ผ่านมา ผู้เขียนได้กล่าวถึงอัลกอริทึม LZSS พร้อมทั้งได้อธิบายถึงซอร์สโค้ดในส่วนของการบีบอัดข้อมูลหรือรูทีน CompressFile ของโปรแกรม LZSS ซึ่งเป็นโปรแกรมตัวอย่างแสดงการบีบอัดข้อมูลด้วยอัลกอริทึม LZSS ส่วนในบทความตอนนี้ก็จะอธิบายต่อเนื่องจากตอนที่ผ่านมา โดยจะอธิบายถึงการทำงานของโค้ดในส่วนของการถอดรหัสข้อมูล (encoding) ซึ่งจะเป็นการทำงานในส่วนของรูทีน ExpandFile ตามที่ได้กล่าวมาแล้วในตอนที่ผ่านมาว่าการถอดรหัสข้อมูลด้วยอัลกอริทึม LZSS จะสามารถทำงานได้เร็วกว่าโค้ดในส่วนของการบีบอัดข้อมูล ดังนั้นในส่วนตัวของผู้เขียนแล้วอยากจะแนะนำให้โปรแกรมเมอร์ที่สนใจได้ทำการพัฒนาโปรแกรมบีบอัดข้อมูลสำเร็จรูปโดยการนำอัลกอริทึม LZSS ไปใช้ หรือถ้าหากโปรแกรมเมอร์ท่านใดที่มีความเข้าใจในการสร้างไฟล์ไลบรารี .DLL สำหรับแอพพลิเคชันที่ทำงานภายใต้ระบบปฏิบัติการวินโดวส์ ก็จะสามารถนำไปสร้างเป็นฟังก์ชันที่ช่วยในการบีบอัดและขยายขนาดข้อมูลได้อย่างดีเลยทีเดียว

แต่อย่างไรก็ตาม เนื่องจากโปรแกรมตัวอย่างยังไม่ใช้โปรแกรมที่สมบูรณ์ และยังคงมีโค้ดในหลาย ๆ ส่วนที่ยังไม่ได้รับการ optimization ดังนั้นถ้าหากผู้อ่านท่านใดที่จะนำแนวคิดจากโปรแกรมตัวอย่างไปพัฒนาต่อในเชิงธุรกิจ ก็ควรที่จะทำการปรับปรุงสมรรถนะของโค้ดให้ดีกว่านี้ครับ

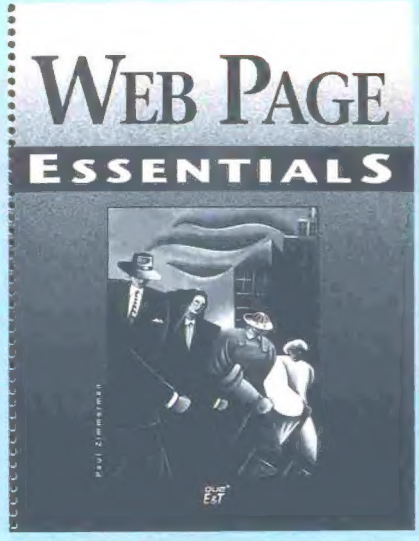
รูทีน ExpandFile

การถอดรหัสข้อมูลด้วยอัลกอริทึม LZSS นั้นสามารถกระทำได้ง่าย ทั้งนี้เนื่องจากในการเข้ารหัสข้อมูลได้ถูกกำหนดให้มีได้ 2 รูปแบบ โดยการนำหน้าแต่ละรูปแบบด้วยบิต 0 หรือ 1 ดังนั้นในการถอดรหัสข้อมูลจึงเพียงแต่เริ่มต้นด้วยการอ่านบิตแรกของแต่ละส่วนข้อมูลที่ถูกบีบอัดเท่านั้น ก็จะทราบถึงรูปแบบในการจัดเก็บ จากนั้นจึงทำการการถอดรหัสข้อมูลตามรูปแบบนั้น ๆ และในส่วนสุดท้ายเมื่อทำการถอดรหัสข้อมูลแล้ว ก็ต้องทำการอัพเดตข้อมูลลงในหน้าต่างข้อความ (text window) และทำการบันทึกข้อมูลที่ถอดรหัสลงในไฟล์ปลายทาง ซึ่งขบวนการถอดรหัสนี้จะดำเนินไปเรื่อย ๆ จนกระทั่งพบค่าของ END_OF_STREAM จากไฟล์บีบอัดข้อมูล จึงถือได้ว่าเป็นจุดสิ้นสุดของการถอดรหัสข้อมูล โดยที่โค้ดของรูทีน ExpandFile ก็มีดังนี้

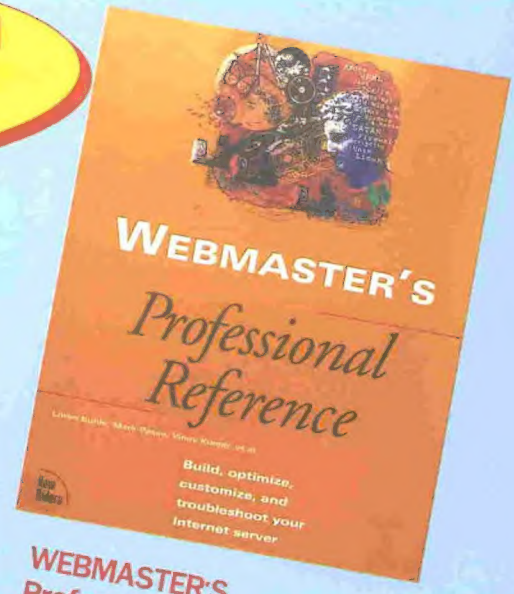
หนังสือต่างประเทศ
จัดจำหน่ายโดยซีเอ็ด



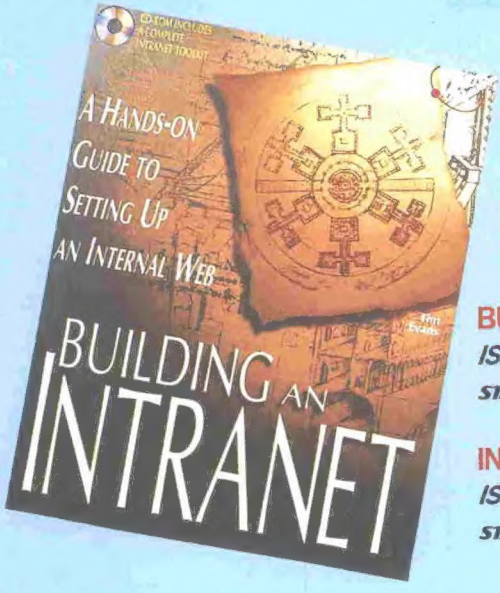
WEB SITE CONSTRUCTION KIT
ISBN : 1-57521-079-7
ราคา 1,984 บาท



WEB PAGE ESSENTIALS
ISBN : 1-57576-368-0
ราคา 557 บาท

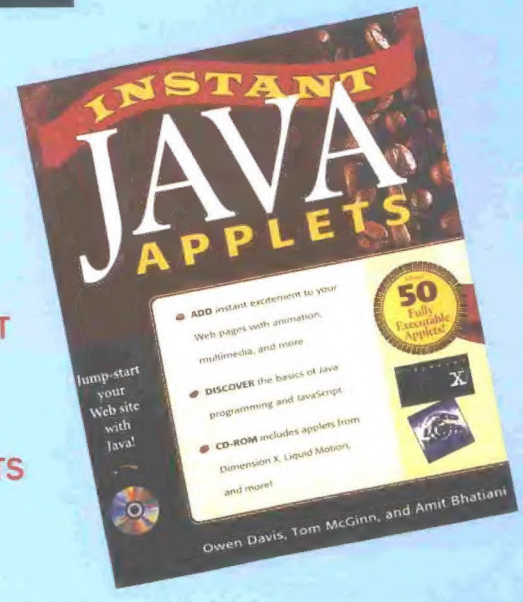


WEBMASTER'S Professional Reference
ISBN : 1-56205-473-2
ราคา 1,364 บาท



BUILDING AN INTRANET
ISBN : 1-57521-071-1
ราคา 1,364 บาท

INSTANT JAVA APPLETS
ISBN : 1-56276-386-5
ราคา 744 บาท



ใบสั่งซื้อหนังสือทางไปรษณีย์

◆ ข้าพเจ้าชื่อ/นามสกุล .....ที่อยู่ .....ซอย .....

ถนน .....แขวง/ตำบล .....เขต/อำเภอ .....จังหวัด .....รหัสไปรษณีย์ .....

มีความประสงค์สั่งซื้อหนังสือทางไปรษณีย์ ตามรหัสและชื่อหนังสือดังต่อไปนี้

1. รหัสสั่งซื้อ .....ชื่อหนังสือ .....ราคา ..... บาท
2. รหัสสั่งซื้อ .....ชื่อหนังสือ .....ราคา ..... บาท
3. รหัสสั่งซื้อ .....ชื่อหนังสือ .....ราคา ..... บาท
4. รหัสสั่งซื้อ .....ชื่อหนังสือ .....ราคา ..... บาท
5. รหัสสั่งซื้อ .....ชื่อหนังสือ .....ราคา ..... บาท

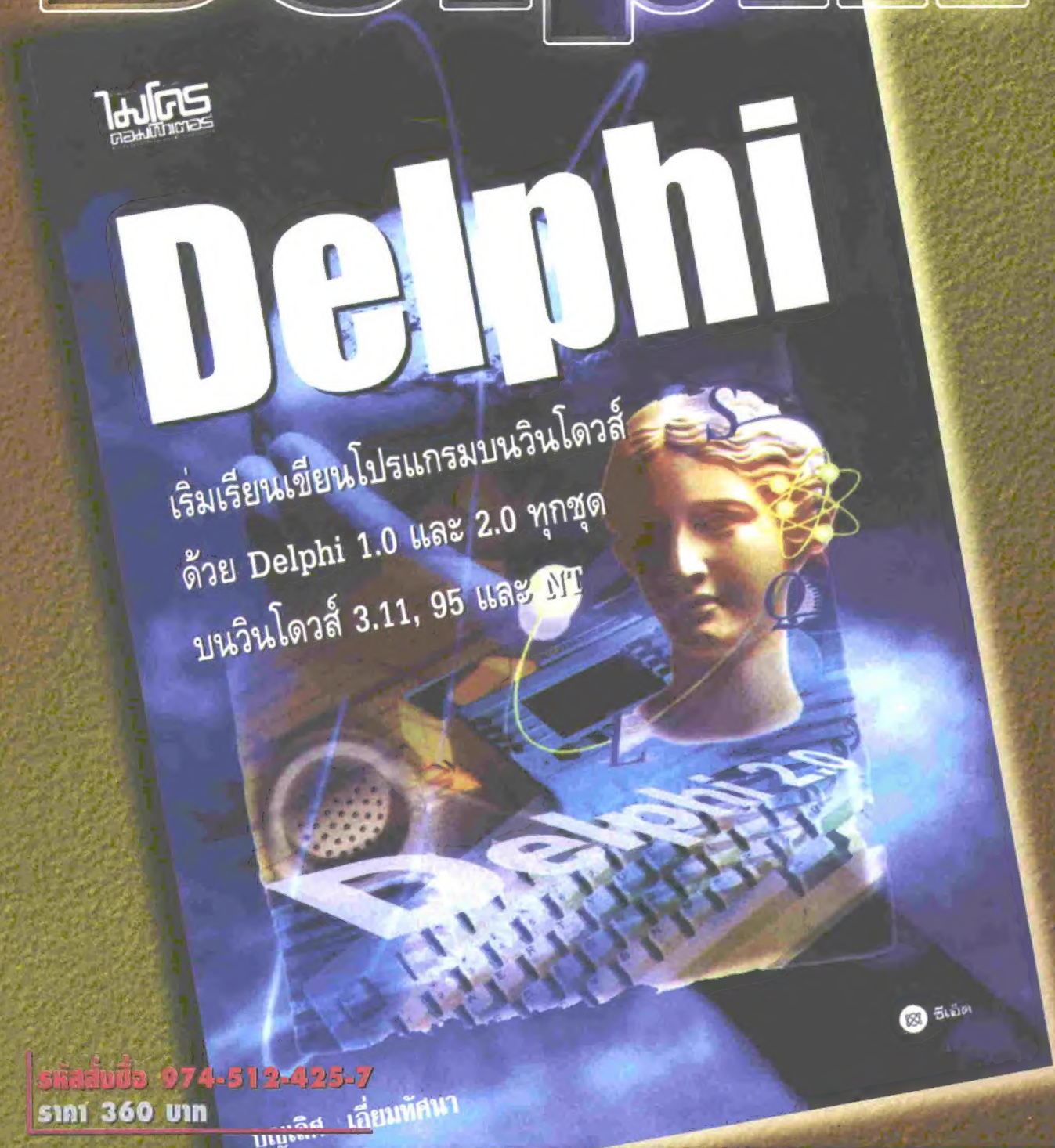
รวมเป็นเงิน .....บาท พร้อมกันนี้ข้าพเจ้าได้แนบการชำระเงินเป็น ☐ ตั๋วแลกเงิน ☐ ธนาณัติ สั่งจ่ายปท.ดินแดง

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน) 800/43-45 ซอยตระกูลสุข ถนนอโศก-ดินแดง เขตดินแดง กรุงเทพฯ 10320 โทร. 248-6280, 642-9800 โทรสาร (02) 246-5999

พิเศษ สมาชิกวารสารซีเอ็ดทุกฉบับ ลด 10% เพิ่มค่าลงทะเบียน 4 บาท ค่าจัดส่งฟรี โดยระบุหมายเลขสมาชิก

หมายเหตุ : หากท่านไม่ต้องการตัดหน้านี้ออกจากเล่ม ท่านจะทำสำเนาหรือคัดลอกข้อความส่งมาแทนก็ได้

Delphi



รหัสสั่งซื้อ 974-512-425-7
ราคา 360 บาท

บุญเลิศ เอี่ยมทัศนา

Delphi

มิติใหม่ของการเขียนโปรแกรมบนวินโดวส์ ด้วย Delphi 1.0 และ 2.0 ทุกชุด สำหรับวินโดวส์ 3.11, 95 และ NT ง่ายเพียงปลายนิ้วสัมผัส ลดเวลาของการพัฒนาโปรแกรม เพียบพร้อมด้วยคอมพายเลอร์ภาษาปาสคาล เอดิเตอร์ และยูทิลิตี้ต่าง ๆ พร้อมคำอธิบายเนื้อหาชัดเจนทุกขั้นตอน แม้แต่ผู้ที่ไม่เคยเขียนโปรแกรมมาก่อนก็สามารถเรียนรู้จากตัวอย่างได้ด้วยตนเอง

```
current_position = 1;
   for ( ; ; ) {
      if ( InputBit( input ) ) {
         c = (int) InputBits( input, 8 );
         putc( c, output );
         window[ current_position ] = (unsigned char) c;
         current_position = MOD_WINDOW( current_position + 1 );
      } else {
         match_position = (int) InputBits( input, INDEX_BIT_COUNT );
         if ( match_position == END_OF_STREAM )
            break;
         match_length = (int) InputBits( input, LENGTH_BIT_COUNT );
         match_length += BREAK_EVEN;
         for ( i = 0 ; i <= match_length ; i++ ) {
            c = window[ MOD_WINDOW( match_position + i ) ];
            putc( c, output );
            window[ current_position ] = (unsigned char) c;
            current_position = MOD_WINDOW( current_position + 1 );
         }
      }
```

ถ้าหากขนาดของหน้าต่างข้อความโตเกินไป ก็อาจจะทำให้โปรแกรมทำงานได้ช้าลงก็ได้ ทั้งนี้เนื่องจากการมีหน้าต่างข้อความขนาดโต ๆ ก็ย่อมหมายถึงการมีแผนภูมิต้นไม้ที่โตตามไปด้วย ซึ่งจากบทความตอนที่แล้วผู้อ่านก็จะทราบว่า ขั้นตอนที่ช้าที่สุดในการเข้ารหัสข้อมูลก็คือการอัพเดตข้อมูล (insertion หรือ deletion) ภายในแผนภูมิต้นไม้ ซึ่งในกรณีที่เลวร้ายที่สุดก็คือไม่มีข้อความใด ๆ เลยในหน้าต่างข้อความที่ซ้ำกับข้อมูลใน look ahead buffer ก็จะทำให้การค้นหาและอัพเดตข้อมูลภายในแผนภูมิต้นไม้สูญเปล่า

และก่อนที่จะจบบทความ ผู้เขียนก็จะขอแนะนำแนวคิดที่ผู้อ่านสามารถนำไปใช้ในการปรับปรุงโปรแกรม LZSS ให้ดีขึ้นได้ต่อไป โดยให้สังเกตที่ค่า (จำนวนบิต) ของตัวแปรครอบคลุม INDEX_BIT_COUNT และ LENGTH_BIT_COUNT ซึ่งใช้ในการกำหนดขนาดของหน้าต่างข้อความและความยาวของโค้ดที่เปรียบเทียบ โดยถ้าหากเราเพิ่มค่าของตัวแปรทั้งสอง ก็หมายถึงการเพิ่มขนาดของหน้าต่างข้อความ ซึ่งจะเกิดผลดีตรงที่ลดจำนวนครั้งในการอ่านข้อมูลจากไฟล์ และจะส่งผลต่อความเร็วของโปรแกรมอีกด้วย แต่ในทางกลับกัน ดังนั้นการที่จะกำหนดขนาดของหน้าต่างข้อความที่เหมาะสมนั้นจึงขึ้นกับการทดลองของผู้อ่าน โดยการทดลองเข้ารหัสและถอดรหัสข้อมูลจากไฟล์หลาย ๆ ขนาดที่แตกต่างกัน และหาค่าเฉลี่ยของเวลาที่ได้จากทั้ง 2 ขบวนการ เพื่อสรุปถึงจุดที่เหมาะสมที่สุด ส่วนอีกจุดหนึ่งที่น่าสนใจที่จะนำมาปรับปรุงโปรแกรม LZSS ก็คือการกำจัดข้อความที่ซ้ำกันในหน้าต่างข้อความ ซึ่งจะช่วยในการจัดสรรการใช้พื้นที่ของหน้าต่างข้อความได้อย่างมีประสิทธิภาพอีกด้วย พบกันใหม่ฉบับหน้าสวัสดีครับ

```
/*
 * LZSSMAIN.H
 */

#ifndef _LZSSMAIN_H
#define _LZSSMAIN_H

void CompressFile( FILE *, BIT_FILE *, int , char *[] );
void ExpandFile( BIT_FILE *, FILE *, int , char *[] );

extern char *Usage;
extern char *CompressionName;

#endif  /* _LZSSMAIN_H */
```

ลิสติงที่ 1
ไฟล์ LZSSMAIN.H

```
/*
 * LZSSMAIN.C After this is built into a compression program of any sort,
 *            the program can be called like this:
 *
 *       lzss  infile  outfile  option
 *       infile          Input filename.
 *       outfile         Output filename.
 *       option -c       Compress data file.
 *              -e       Expand data file.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bitio.h"
#include "lzssmain.h"
```

ลิสติงที่ 2
ไฟล์ LZSSMAIN.C

ลิสติงที่ 2 (ต่อ)

```
void usage_exit(char *);
void print_ratios(char *, char *);
long file_size(char *);

int main( int argc, char *argv[] )
{
    setbuf( stdout, NULL );
    if ( argc < 3 )
       usage_exit( argv[ 0 ] );

    if ( strcmp( argv[ 3 ], "-c" ) == 0 ) {
       BIT_FILE *output;
       FILE *input;

       input = fopen( argv[ 1 ], "rb" );
       if ( input == NULL )
            fatal_error( "Error opening %s for input\n", argv[ 1 ] );

         output = OpenOutputBitFile( argv[ 2 ] );

         if ( output == NULL )
              fatal_error( "Error opening %s for output\n", argv[ 2 ] );

         printf( "\nCompressing %s to %s\n", argv[ 1 ], argv[ 2 ] );
       printf( "Using %s\n", CompressionName );
       CompressFile( input, output, argc, argv );
       CloseOutputBitFile( output );
       fclose( input );
       print_ratios( argv[ 1 ], argv[ 2 ] );
       return( 0 );
    } else if ( strcmp( argv[ 3 ], "-e" ) == 0 ) {
       FILE *output;
       BIT_FILE *input;

       input = OpenInputBitFile( argv[ 1 ] );

         if ( input == NULL )
              fatal_error( "Error opening %s for input\n", argv[ 1 ] );

         output = fopen( argv[ 2 ], "wb" );

         if ( output == NULL )
              fatal_error( "Error opening %s for output\n", argv[ 2 ] );

         printf( "\nExpanding %s to %s\n", argv[ 1 ], argv[ 2 ] );
       printf( "Using %s\n", CompressionName );
       ExpandFile( input, output, argc, argv );
       CloseInputBitFile( input );
       fclose( output );
       putc( '\n', stdout );
       return( 0 );
    } else
       usage_exit( argv[ 0 ] );
}

void usage_exit( char *prog_name )
{
    char *short_name;
    char *extension;

    short_name = strrchr( prog_name, '\\' );
    if ( short_name == NULL )
        short_name = strrchr( prog_name, '/' );
    if ( short_name == NULL )
        short_name = strrchr( prog_name, ':' );
    if ( short_name != NULL )
        short_name++;
    else
        short_name = prog_name;

      extension = strrchr( short_name, '.' );
    if ( extension != NULL )
        *extension = '\0';
```

→

ลิสติงที่ 2 (ต่อ)

```
        printf( "\nUsage:  %s %s\n", short_name, Usage );
    exit( 0 );
}

#ifndef SEEK_END
   #define SEEK_END 2
#endif

long file_size( char *name )
{
    long eof_ftell;
    FILE *file;

    file = fopen( name, "r" );
    if ( file == NULL )
        return( 0L );

      fseek( file, 0L, SEEK_END );
    eof_ftell = ftell( file );
    fclose( file );
    return( eof_ftell );
}

void print_ratios( char *input, char *output )
{
    long input_size;
    long output_size;
    int ratio;

    input_size = file_size( input );
    if ( input_size == 0 )
        input_size = 1;

      output_size = file_size( output );
    ratio = 100 - (int) ( output_size * 100L / input_size );
    printf( "\nInput bytes:        %ld\n", input_size );
    printf( "Output bytes:       %ld\n", output_size );
    if ( output_size == 0 )
        output_size = 1;

      printf( "Compression ratio:  %d%%\n", ratio );
}
```

ลิสติงที่ 3
ไฟล์ LZSS.C

```
/* LZSS.C
 * This is the LZSS module, which implements an LZ77 style compression algorithm.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "bitio.h"

/*
 * - INDEX_BIT_COUNT tells how many bits we allocate to indices into the
 *   text window.  This directly determines the WINDOW_SIZE.
 * - LENGTH_BIT_COUNT tells how many bits we allocate for the length of
 *   an encode phrase. This determines the size of the look ahead buffer.
 * - TREE_ROOT is a special node in the tree that always points to
 *   the root node of the binary phrase tree.
 * - MOD_WINDOW() is a macro used to perform arithmetic on tree indices.
 */

#define INDEX_BIT_COUNT          12
#define LENGTH_BIT_COUNT         4
#define WINDOW_SIZE              ( 1 << INDEX_BIT_COUNT )
#define RAW_LOOK_AHEAD_SIZE      ( 1 << LENGTH_BIT_COUNT )
#define BREAK_EVEN               ( ( 1 + INDEX_BIT_COUNT + LENGTH_BIT_COUNT ) / 9 )
#define LOOK_AHEAD_SIZE          ( RAW_LOOK_AHEAD_SIZE + BREAK_EVEN )
#define TREE_ROOT                WINDOW_SIZE
#define END_OF_STREAM            0
#define UNUSED                   0
#define MOD_WINDOW( a )          ( ( a ) & ( WINDOW_SIZE - 1 ) )
```

ลิสติงที่ 3 (ต่อ)

```
char *CompressionName = "LZSS Algorithm";
char *Usage           = "in-file out-file -c/-e \n\n";

/*
 * These are the two global data structures used in this program.
 * The window[] array is exactly that, the window of previously seen text,
 * as well as the current look ahead text.  The tree[] structure contains
 * the binary tree of all of the strings in the window sorted in order.
 */

unsigned char window[ WINDOW_SIZE ];

struct {
    int parent;
    int smaller_child;
    int larger_child;
} tree[ WINDOW_SIZE + 1 ];

/*
 * Function prototypes for K&R brethren.
 */

void InitTree( int );
void ContractNode( int , int );
void ReplaceNode( int , int);
int FindNextNode( int );
void DeleteString( int );
int AddString( int , int *);
void CompressFile( FILE *, BIT_FILE *, int , char *[] );
void ExpandFile( BIT_FILE *, FILE *, int , char *[] );


void InitTree( int r )
{
    tree[ TREE_ROOT ].larger_child = r;
    tree[ r ].parent = TREE_ROOT;
    tree[ r ].larger_child = UNUSED;
    tree[ r ].smaller_child = UNUSED;
}

void ContractNode( int old_node, int new_node )
{
       int old_parent;

       old_parent = tree[ old_node ].parent;
       tree[ new_node ].parent = old_parent;
       if ( tree[ old_parent ].larger_child == old_node )
              tree[ old_parent ].larger_child = new_node;
    else
              tree[ old_parent ].smaller_child = new_node;
    tree[ old_node ].parent = UNUSED;
}

void ReplaceNode( int old_node, int new_node )
{
    int parent;

    parent = tree[ old_node ].parent;
    if ( tree[ parent ].smaller_child == old_node )
        tree[ parent ].smaller_child = new_node;
    else
        tree[ parent ].larger_child = new_node;

       tree[ new_node ] = tree[ old_node ];
    tree[ tree[ new_node ].smaller_child ].parent = new_node;
    tree[ tree[ new_node ].larger_child ].parent = new_node;
    tree[ old_node ].parent = UNUSED;
}

int FindNextNode( int node )
{
    int next;
```

→

ลิสติงที่ 3 (ต่อ)

```
    next = tree[ node ].smaller_child;
    while ( tree[ next ].larger_child != UNUSED )
        next = tree[ next ].larger_child;
    return( next );
}

void DeleteString( int p )
{
    int  replacement;

    if ( tree[ p ].parent == UNUSED )
        return;
    if ( tree[ p ].larger_child == UNUSED )
        ContractNode( p, tree[ p ].smaller_child );
    else if ( tree[ p ].smaller_child == UNUSED )
        ContractNode( p, tree[ p ].larger_child );
    else {
        replacement = FindNextNode( p );
        DeleteString( replacement );
        ReplaceNode( p, replacement );
    }
}

int AddString( int new_node, int *match_position )
{
    int i;
    int test_node;
    int delta;
    int match_length;
    int *child;

    if ( new_node == END_OF_STREAM )
        return( 0 );
    test_node = tree[ TREE_ROOT ].larger_child;
    match_length = 0;
    for ( ; ; ) {
        for ( i = 0 ; i < LOOK_AHEAD_SIZE ; i++ ) {
            delta = window[ MOD_WINDOW( new_node + i ) ] -
                    window[ MOD_WINDOW( test_node + i ) ];
            if ( delta != 0 )
                break;
        }
        if ( i >= match_length ) {
            match_length = i;
            *match_position = test_node;
            if ( match_length >= LOOK_AHEAD_SIZE ) {
                ReplaceNode( test_node, new_node );
                return( match_length );
            }
        }
        if ( delta >= 0 )
            child = &tree[ test_node ].larger_child;
        else
            child = &tree[ test_node ].smaller_child;
        if ( *child == UNUSED ) {
            *child = new_node;
            tree[ new_node ].parent = test_node;
            tree[ new_node ].larger_child = UNUSED;
            tree[ new_node ].smaller_child = UNUSED;
            return( match_length );
        }
        test_node = *child;
    }
}

void CompressFile( FILE *input, BIT_FILE *output, int argc, char *argv[] )
{
    int i;
    int c;
    int look_ahead_bytes;
    int current_position;
    int replace_count;
    int match_length;
    int match_position;
```

→

ลิสติงที่ 3 (ต่อ)

```
    current_position = 1;
    for ( i = 0 ; i < LOOK_AHEAD_SIZE ; i++ ) {
        if ( ( c = getc( input ) ) == EOF )
            break;
        window[ current_position + i ] = (unsigned char) c;
    }
    look_ahead_bytes = i;
    InitTree( current_position );
    match_length = 0;
    match_position = 0;
    while ( look_ahead_bytes > 0 ) {
        if ( match_length > look_ahead_bytes )
            match_length = look_ahead_bytes;
        if ( match_length <= BREAK_EVEN ) {
            replace_count = 1;
            OutputBit( output, 1 );
            OutputBits( output,
                        (unsigned long) window[ current_position ], 8 );
        } else {
            OutputBit( output, 0 );
            OutputBits( output,
                        (unsigned long) match_position, INDEX_BIT_COUNT );
            OutputBits( output,
                        (unsigned long) ( match_length - ( BREAK_EVEN + 1 ) ),
                        LENGTH_BIT_COUNT );
            replace_count = match_length;
        }
        for ( i = 0 ; i < replace_count ; i++ ) {
            DeleteString( MOD_WINDOW( current_position + LOOK_AHEAD_SIZE ) );
            if ( ( c = getc( input ) ) == EOF )
                look_ahead_bytes--;
            else
                window[ MOD_WINDOW( current_position + LOOK_AHEAD_SIZE ) ]
                        = (unsigned char) c;
            current_position = MOD_WINDOW( current_position + 1 );
            if ( look_ahead_bytes )
                match_length = AddString( current_position, &match_position );
        }
    };
    OutputBit( output, 0 );
    OutputBits( output, (unsigned long) END_OF_STREAM, INDEX_BIT_COUNT );
}

void ExpandFile( BIT_FILE *input, FILE *output, int argc, char *argv[] )
{
    int i;
    int current_position;
    int c;
    int match_length;
    int match_position;

    current_position = 1;
    for ( ; ; ) {
        if ( InputBit( input ) ) {
            c = (int) InputBits( input, 8 );
            putc( c, output );
            window[ current_position ] = (unsigned char) c;
            current_position = MOD_WINDOW( current_position + 1 );
        } else {
            match_position = (int) InputBits( input, INDEX_BIT_COUNT );
            if ( match_position == END_OF_STREAM )
                break;
            match_length = (int) InputBits( input, LENGTH_BIT_COUNT );
            match_length += BREAK_EVEN;
            for ( i = 0 ; i <= match_length ; i++ ) {
                c = window[ MOD_WINDOW( match_position + i ) ];
                putc( c, output );
                window[ current_position ] = (unsigned char) c;
                current_position = MOD_WINDOW( current_position + 1 );
            }
        }
    }
}
```